

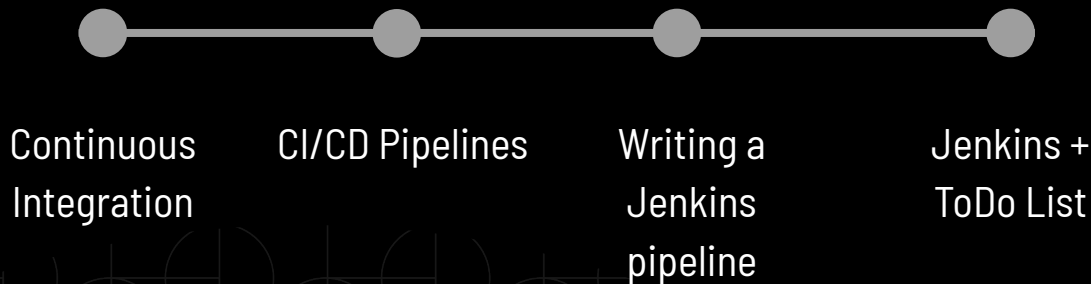
Modern Engineering

March 14 | Day 14

Prudential Financial



LESSON ROADMAP



MEF MODULE 5 DAY 14: CI/CD Pipelines with Jenkins

Schedule	
9:00–9:15 am	Welcome and Warm-Up
9:15–10:00 am	Continuous Integration and Github
10:00 am–12:30 pm	Jenkins CI/CD Walkthrough
12:30–1:30 pm	Lunch
1:30–4:50 pm	Open Lab and Review Time
4:50–5:00 pm	Bring It Home

LEARNING OBJECTIVES

1

Explain the relationship between git and continuous integration/continuous deployment (CI/CD)

2

Build a functional CI/CD pipeline using Jenkins

3

Explain the benefits of CI/CD in the Software Development Cycle

4

Apply CI/CD to automated testing and deployment in an Express application



Modern Engineering

Continuous Integration and Git



CI/CD

Continuous integration (CI) and **continuous delivery** (CD) are used to build, test, and deploy software quickly and safely, ultimately delivering value to your users more quickly.

Generally, companies begin with **continuous integration**.

Continuous delivery goes a step further than CI by automating the deployment of releases. This includes infrastructure and configuration changes.

Continuous deployment goes a step further than continuous delivery by deploying every change to production automatically.

Commit Small Changes Often

It is best practice for developers to commit to the mainline branch *often*.

- Changes should be as small as possible
- ***If the build breaks***, anyone can quickly pinpoint the specific issue

With many changes happening all the time, a fast build process is critical, so developers can get feedback about broken builds right away.

Quick Question:

How ***often*** do you think a developer should be pushing their changes to the mainline branch?



Come off of mute or type your answer in the chat!



Discussion:

Why is CI/CD Important?

10 Minutes



Consider how *often* we make changes to the mainline branch in CI/CD.

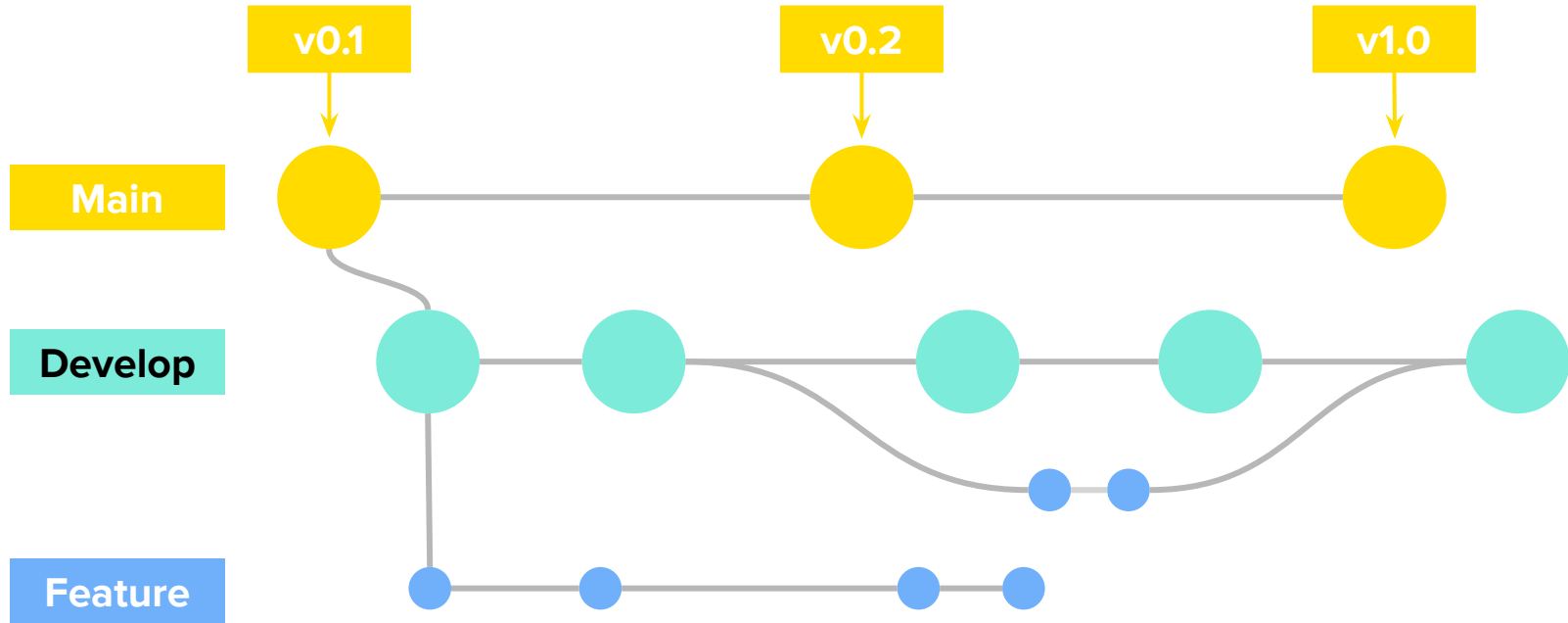
What would be the biggest issue if we were making
larger changes, less often?

What might be the overall impact to build and development process?

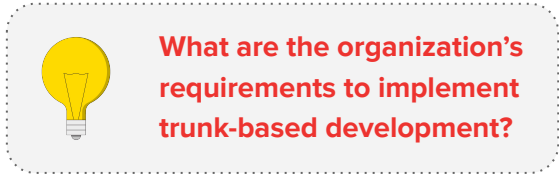


Come off mute
and discuss!

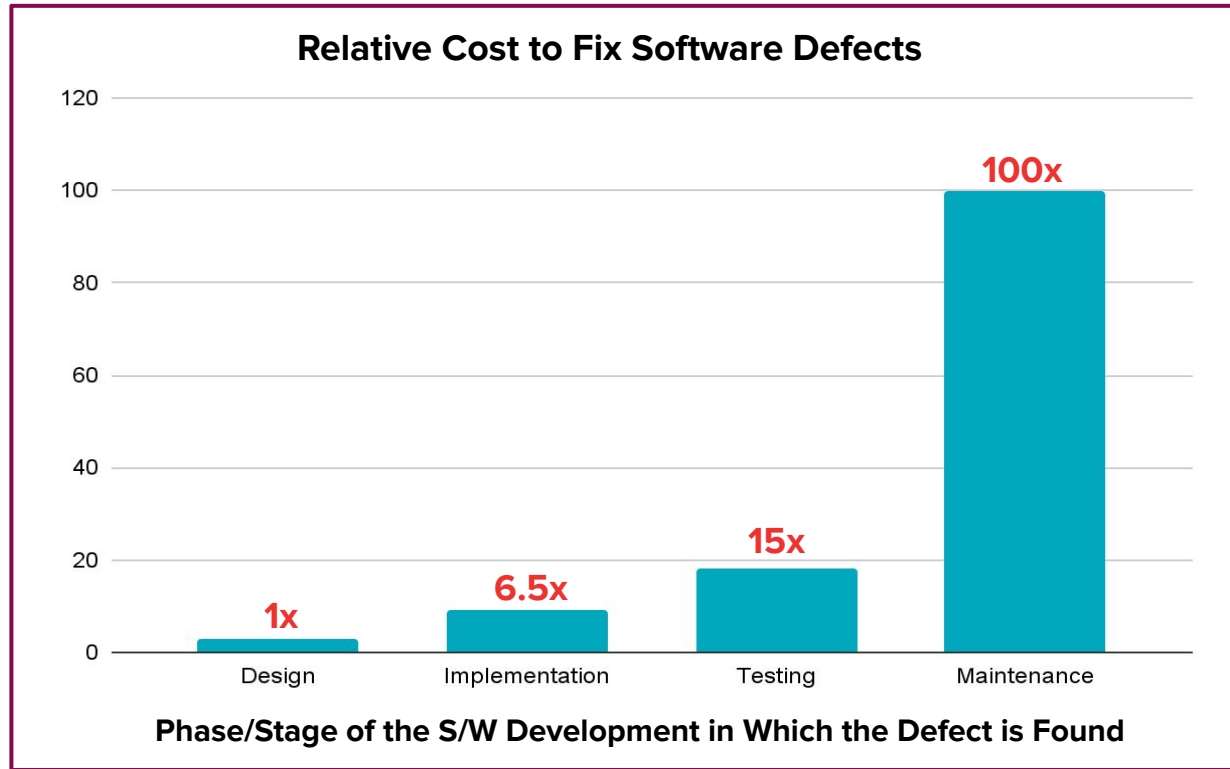
Feature Branches



Example of GIT Workflow



Visualized





Real Cases:

Why is CI/CD Important?

SAMSUNG

This concept is fundamental to why we do CI/CD. Let us discuss a real-world example:

Ever heard of the Samsung Note 7 fiasco? Note 7 phones had a faulty battery management system that caused them to catch on fire. They would have saved so much money if they had caught the bug in the early stages of development.

Can you guess?

How much do you think this particular bug ended up costing Samsung?





Real Cases:

Why is CI/CD Important?

SAMSUNG

**This bug cost Samsung nearly
\$17 billion!**

If it had been caught it earlier, they could have saved their reputation and many headaches.

This is why CI/CD is important.



— Continuous Delivery

Continuous Delivery

Continuous delivery (CD) is an approach that builds on continuous integration. CD ensures that our code is always in a deployable state, even with thousands of developers making changes regularly. Teams produce software in short cycles so they can release the software whenever necessary. CD helps teams save time and money and makes delivering changes less risky.



Development, Staging, and Production Environments

In most enterprises, there are several different environments where a new version of the application (a release candidate) needs to be built, tested, and/or reviewed before deployment to the production environment.

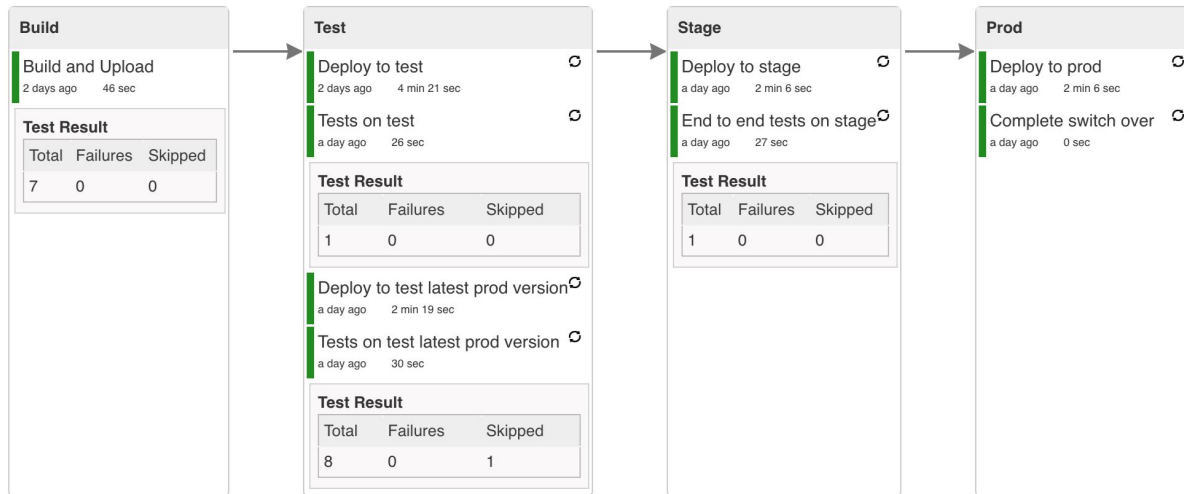
1. **Build**: Oftentimes, you'll see a build environment where the code is compiled and application artifacts are constructed (e.g., an rpm, war file, or image). Sometimes, unit tests will run here, too. Once built, the artifact is pushed to a repository and should never change.
2. **Test**: The artifact would then be pulled from the repository and installed in a test environment for integration testing, performance testing, and other testing purposes.
3. **Stage**: A staging environment is the last environment before promoting the artifact to production. That's where more tests can be run (automated or not) and/or human users may perform some kind of acceptance testing.

Development, Staging, and Production Environments

In CI/CD, this whole process — including deployments from build to test to stage and (optionally) to prod — should be automated by your build pipeline. Here's an example of what that pipeline might look like in Jenkins:

1.0.0.M1-161008_221111-VERSION triggered by user anonymous started 2 days ago

Total build time: 13 min 3 sec



The Other CD

In some companies, there will need to be a final approval or review by a human before software can actually be deployed to production. If that isn't the case, you can also fully automate that final deployment step (a practice known as continuous deployment).

Continuous delivery ensures that code can be rapidly and safely deployed to production and that applications function as expected through rigorous automated testing.

Continuous deployment is the next step of continuous delivery: Every change that passes the automated tests is deployed to production automatically.



Discussion:

Continuous Deployment



Knowledge check: When is continuous deployment helpful? When might you not want to use continuous deployment?

CI/CD



Build Pipelines



How do we catch bugs early?

You catch bugs by **testing early** and **often**.

Most modern software development teams run tests automatically when there is any kind of change to the code.

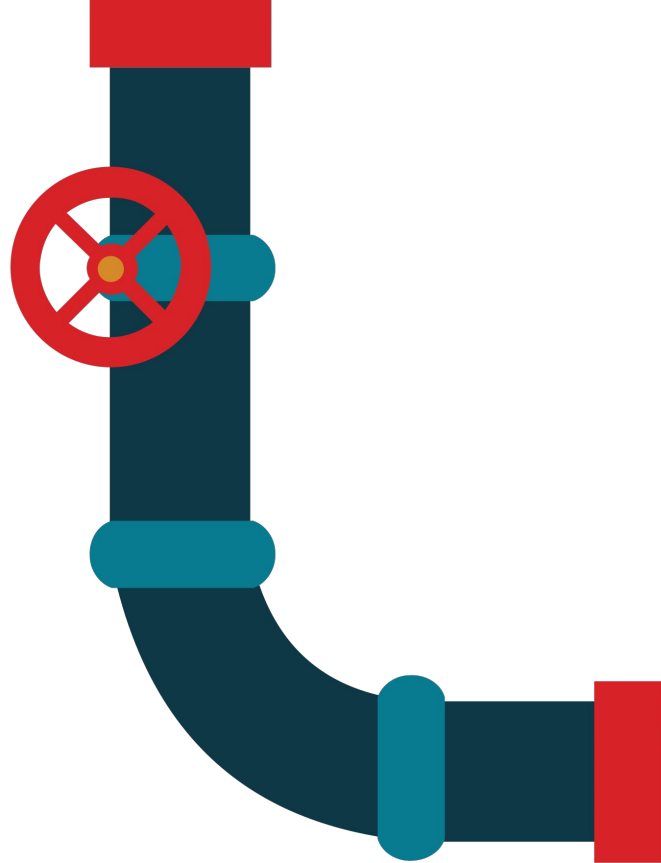


Build Pipeline

A **build pipeline** is a set of automated processes that work together to build, test, and deploy software applications.

When a code change is detected, a build pipeline can build the application and test it.

Pipelines can have stages, and each stage can have steps. These steps can be configured to run your commands and shell scripts.



Characteristics of a build pipeline

Some features we want for our build pipelines:

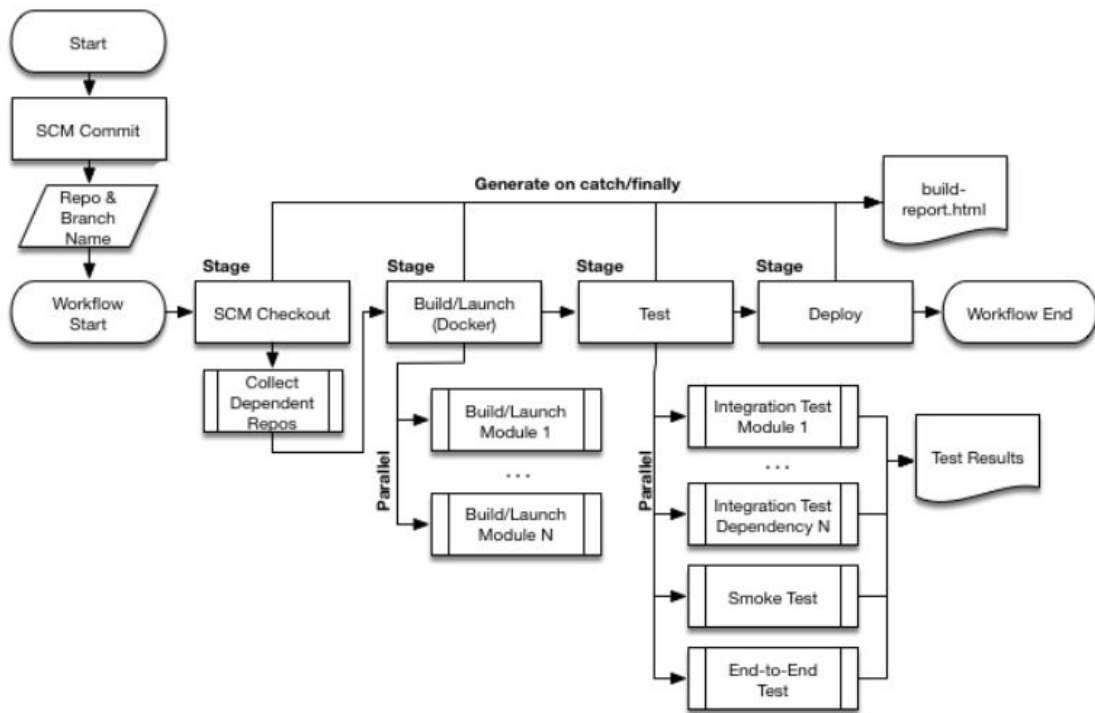
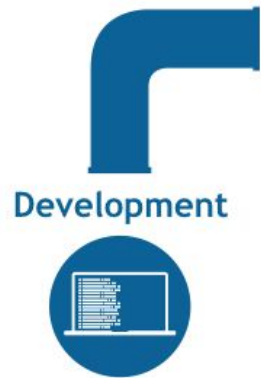


Fast feedback

Repeatable

Reliable

The Deployment Pipeline





Break Time

Jenkins CI/CD



Jenkins



Jenkins



Jenkins

Jenkins is an **open source automation server** used for building, testing, and deploying applications.

Jenkins is a tool built to enable **continuous integration**.

It's the most widely used tool for CI/CD in the industry.

It's highly extensible, supporting a wide range of **plugins** for customization.

Important to note, it also supports the creation of **complex build pipelines**.

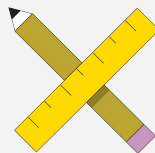




Let's do some Jenkins research. With a partner, take a look at the [Jenkins documentation](#) (or other Jenkins resources).

Answer the following questions:

- How does a Jenkins pipeline work?
- What is a Jenkinsfile?
- What's the difference between a **declarative** and **scripted** Jenkins pipeline?

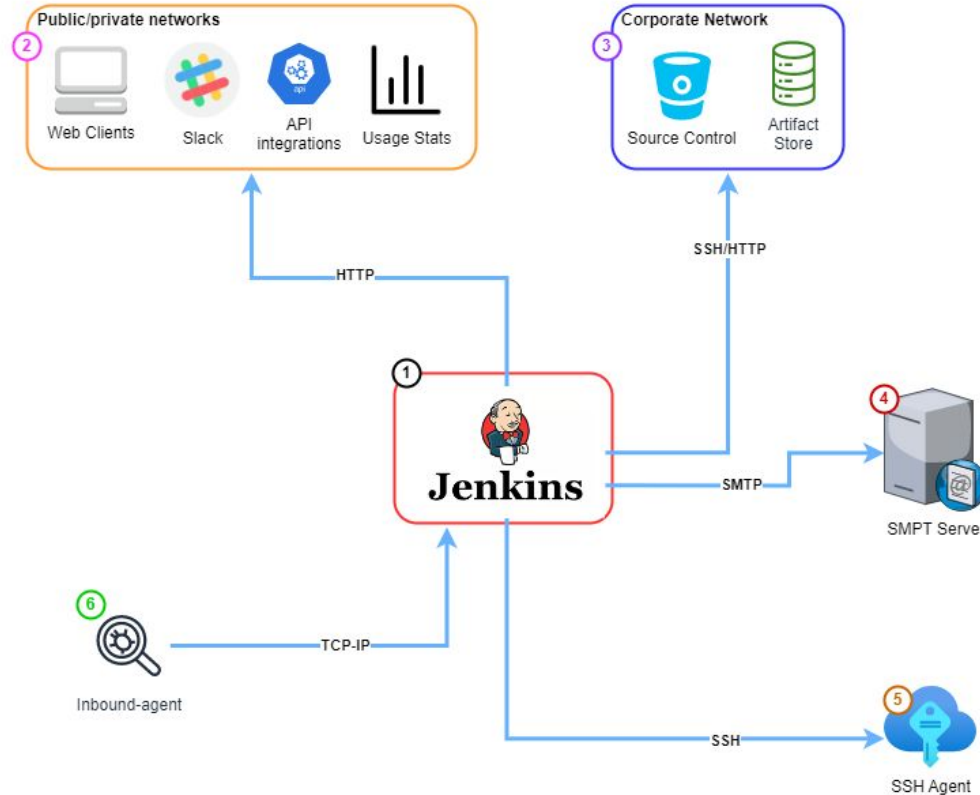


After researching, be prepared to share out your answers.

Jenkins high level architecture



Jenkins



Jenkins Plugins



Jenkins

A core feature of Jenkins are **plugins**, which allow you to extend Jenkins' basic functionality.

Some common types of plugins:

- Source Code Management (SCM) Plugins
- Build Tool Plugins
- Testing Plugins
- Deployment Plugins
- Monitoring Plugins
- Code Quality Plugins



How to I build my pipelines?

One way to build your pipeline in jenkins is to build your pipelines inside of **Jenkinsfiles**.

Pipelines allow for **more complex configuration** and **git versioning**!

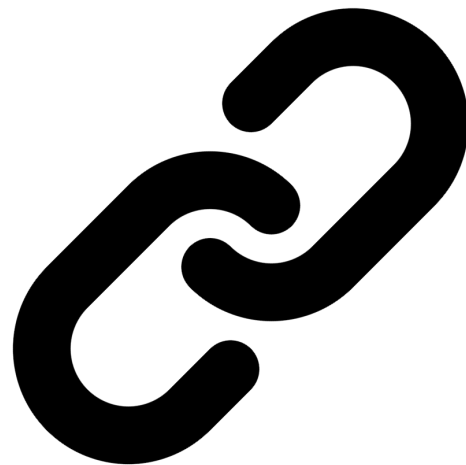
```
pipeline {
  agent {
    docker {
      image 'node:lts-bullseye-slim'
      args '-p 3000:3000'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'npm install'
      }
    }
  }
}
```

Repositories and Nexus

Once your application is built, your **application artifact** will be deployed to a repository such as **Nexus**.

By uploading the artifact to Nexus, the next stages in the pipeline can actually run on different servers and pull the artifact from the repo to operate on it.

Nexus also offers other features for dependency management and audit controls.





Jenkins and Docker Instructor Walkthrough

Instructor will walk through Jenkins CI/CD Pipeline no need to code along.



Break Time



Adding Jenkins to Our To-Do App

Together, let's add a Jenkins CI/CD pipeline to our model To-Do Application, including some automated testing commands that ensure a git push will be blocked if the tests are failing.



Solo Exercise:

Add Jenkins to Your API

120 minutes



Take the API you've built and add a Jenkins build pipeline that executes automated testing before allowing a push to the git repository.

This will require a lot of consulting the documentation and research!

Recap



CI/CD

Discussed CI/CD best practices and the overall benefits



Jenkins

Learned how Jenkins can be used in a CI/CD workflow.



Pipelines

Created a local Jenkins server and set-up Jenkins pipelines.



Solo Exercise:

Complete Docker Lab

Remainder of day



For the remainder of today's session, you'll have time to complete or enhance the [Docker Lab](#)

This will be the final evaluated lab before the final project begins soon!





- Please take a moment to give us your feedback after today's course!
- Use the QR code or follow the link: <https://bit.ly/pruMEF-c3>



Scan Me!

